Lab3 Review

NTHU Logic Design Laboratory

2019/10/2

By Prof. Chun-Yi Lee

Outline

1 Code Structure

2 Coding Style

3 Module Relation

4 Useful Module

Outline

Code Structure (1/6)

- Only one DFF circuit in a module
- Divide your design
- Try internal signals to reduce the complexity
- Avoid using “ else if ”

Code Structure (2/6)

- In DFF
  - Use “ <= ”
  - Avoid any computation in DFF circuits

```
6  always@*(posedge clk)begin
7      if (!rst_n) begin
8          s1 <= ini_s1;
9          s2 <= ini_s2;
10     end else begin
11         s1 <= next_s1;
12         s2 <= next_s2;
13     end
14
15 end
```

Code Structure (3/6)

```
17
18      always@(*)begin
19          if (ren == 1'b0) begin
20              next_dout  = 8'd0;
21          end else begin
22              next_dout  = MEM [addr];
23          end
24      end
25
26      always@(*)begin
27          if ({wen, ren} == 2'b10) begin
28              next_MEM  = din;
29          end else begin
30              next_MEM  = MEM [addr];
31          end
32      end
33
```

- In Combinational circuits
  - An always block handles 1 ~ 4 signals
  - You can use multiple always blocks in a module

Code Structure (4/6)

- Internal signal
  - Define some useful signals to reduce the complexity of your code

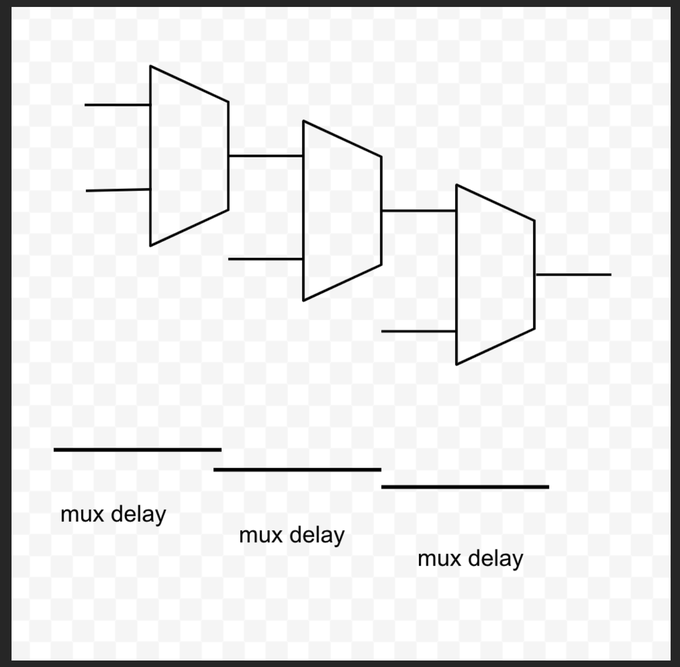
```
 9     parameter up = 1'b0;
10     parameter down = 1'b1;
11
12
13     wire is_border = (out == max || out == min);
14     wire out_of_range = (out > max || out < min);
15     wire is_available = (max > min || enable || !out_of_range);
16
```

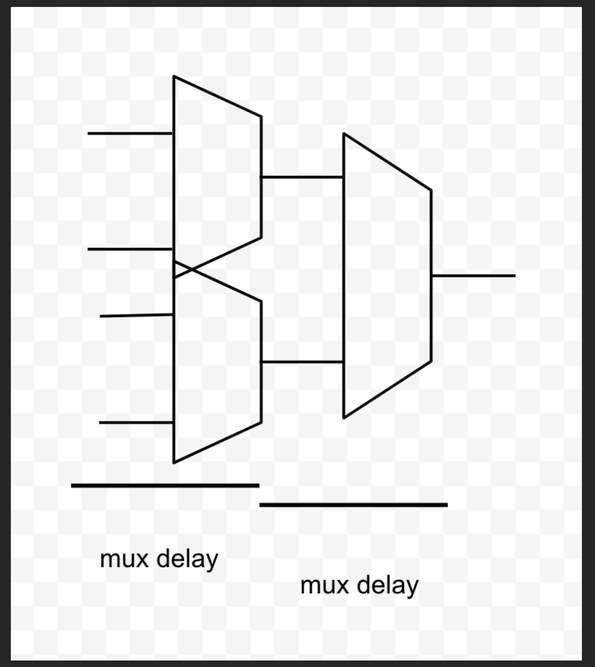
Code Structure (5/6)

```
27      always@(*)begin
28          if (!is_available) begin
29              next_direction = direction;
30              next_out = out;
31          end else begin
32              if (flip) begin
33                  next_direction = !direction;
34                  next_out = (direction == up) ? out - 1'b1 : out + 1'b1;
35              end else begin
36                  if (is_border) begin
37                      next_direction = !direction;
38                      next_out = direction == up) ? out - 1'b1 : out + 1'b1;
39                  end else begin
40                      if (direction == up) begin
41                          next_direction = up;
42                          next_out = out + 1'b1;
43                      end else begin // direction == down
44                          next_direction = down;
45                          next_out = out - 1'b1;
46                      end  // end direction
47                  end // end is_border
```

Code Structure (6/6)

- Avoid using “ else if ” with different condition

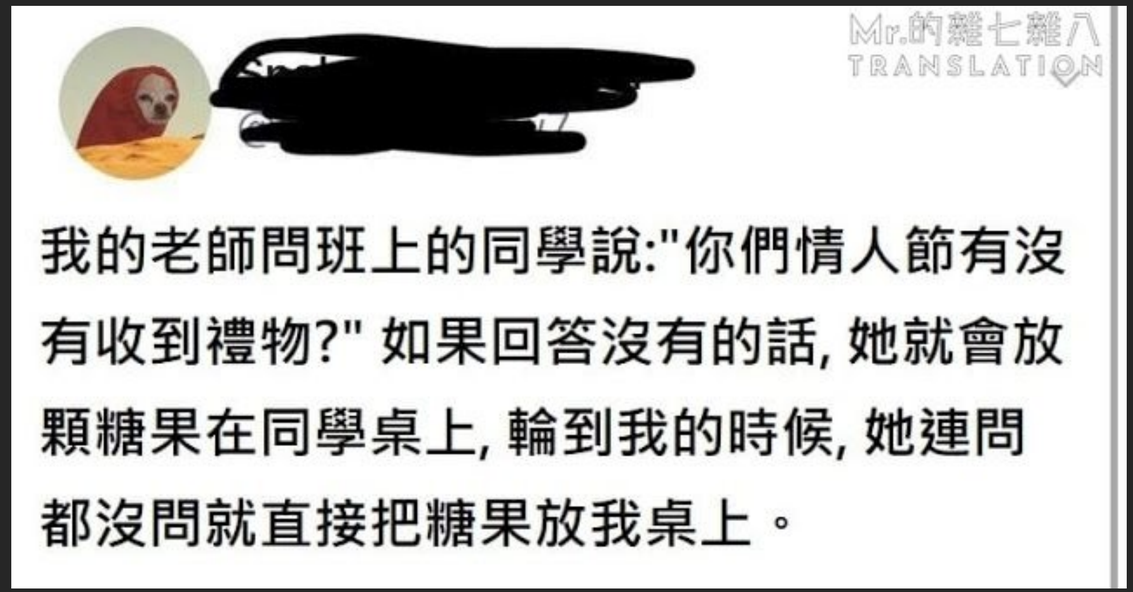




Q & A

Q & A

- 第一題
  - 為什麼要避免使用 `else if` ?



Q & A

```
6  always@*(    1      )begin
7      if (!rst_n) begin
8          s1  2  ini_s1;
9          s2     ini_s2;
10     end else begin
11         s1     next_s1;
12         s2  3  next_s2;
13     end
14
15 end
```

- 第二題

- 填空：)

Outline

1 Code Structure

2 **Coding Style**

3 Module Relation

4 Useful Module

Coding Style (1/4)

- Only RTL is allowed
- No initial block, loop
- Avoid using “ ( ) ? : ” with multiple conditions

Coding Style (2/4)

```
LFSR.v:         d_ff1 <= (rst_n == 0) ? 1'b0 : d_ff0;
LFSR.v:         d_ff2 <= (rst_n == 0) ? 1'b0 : d_ff1;
LFSR.v:         d_ff3 <= (rst_n == 0) ? 1'b1 : d_ff2;
LFSR.v:         d_ff4 <= (rst_n == 0) ? 1'b0 : d_ff3;
_Parameterized_Ping_Pong_Counter.v:    assign nt_cnt = (rst_n == 0) ? min : nt_cnt1;
_Parameterized_Ping_Pong_Counter.v:    assign nt_dirc = (rst_n == 0) ? 1'b0 : nt_dirc1;
_Parameterized_Ping_Pong_Counter_fpga.v:    assign nt_cnt = (rst_n == 0) ? min : nt_cnt1;
_Parameterized_Ping_Pong_Counter_fpga.v:    assign nt_dirc = (rst_n == 0) ? 1'b0 : nt_dirc1;
_Parameterized_Ping_Pong_Counter.v://assign next_direction = (rst_n == 0) ?  `UP : next_direction_1;
_Parameterized_Ping_Pong_Counter.v://assign next_out = (rst_n == 0) ?  min : next_out_1;
_LFSR.v:   assign wback1 = (rst_n == 0)? 1'b0: wback;
_LFSR.v:   assign w01 = (rst_n == 0)? 1'b0: w0;
_LFSR.v:   assign w11 = (rst_n == 0)? 1'b0: w1;
_LFSR.v:   assign w21 = (rst_n == 0)? 1'b1: w2;
_LFSR.v:   assign w31 = (rst_n == 0)? 1'b0: w3;
_Parameterized_Ping_Pong_Counter.v:        assign next_out = (rst_n == 0)? min: next_out1;
_Parameterized_Ping_Pong_Counter.v:        assign next_direction = (rst_n == 0)? 1'b0: next_direction1;
Parameterized_Ping_Pong_Counter.v:    assign next_out = (rst_n == 1'b0)? min: next_out1;
Parameterized_Ping_Pong_Counter.v:    assign next_state = (rst_n == 1'b0)? `INIT: next_state1;
_Parameterized_Ping_Pong_Counter.v:        assign next_state = (rst_n == 1'b0)?`UP : new_state;
_Parameterized_Ping_Pong_Counter.v:        assign next_out = (rst_n == 1'b0)?min :
_Parameterized_Ping_Pong_Counter.v:        assign next_direction = (rst_n == 1'b0)?1'b0:
_Parameterized_Ping_Pong_Counter_fpga.v:   assign next_state = (rst_n == 1'b0)?`UP : new_state;
_Parameterized_Ping_Pong_Counter_fpga.v:   assign next_out = (rst_n == 1'b0)?min :
_Parameterized_Ping_Pong_Counter.v:                        out <= (!rst_n)?min:out;
_Parameterized_Ping_Pong_Counter.v:                        direction <= (!rst_n)?0:direction;
_Parameterized_Ping_Pong_Counter.v:                               out <= (!rst_n)?min:out-1;
_Parameterized_Ping_Pong_Counter.v:                               direction <= (!rst_n)?0:~direction;
_Parameterized_Ping_Pong_Counter.v:                               out <= (!rst_n)?min:((flip)?out-1:out+1);
_Parameterized_Ping_Pong_Counter.v:                               direction <= (!rst_n)?0:((flip)?(~direction):direction);
_Parameterized_Ping_Pong_Counter.v:                               out <= (!rst_n)?min:out+1;
_Parameterized_Ping_Pong_Counter.v:                               direction<=(!rst_n)?0:~direction;
_Parameterized_Ping_Pong_Counter.v:                               out <= (!rst_n)?min:((flip)?out+1:out-1);
_Parameterized_Ping_Pong_Counter.v:                               direction <= (!rst_n)?0:((flip)?(~direction):direction);
```

Coding Style (3/4)

```
24  module DFF(in, out);
25      always@(posedge clk) begin
26          out <= in;
27      end
28  endmodule
29
30  module hahahah();
31      DFF dff0(next_state, state);
32
33      assign next_state  = (!rst_n) ? state_0 : ???;
34  endmodule
35
```

- Not recommended :(

Coding Style (4/4)

- Recommended :)))))))))))

```
module hahahah();
    always@(posedge clk) begin
        if (!rst_n) begin
            state <= state_0;
        end else begin
            state <= next_state;
        end
    end

    always@(*)begin
        case(state)
            state_0: next_state = state_1;
            state_1: next_state = state_2;
            state_2: next_state = state_3;
            state_3: next_state = (conditional) ? state_0 : state_2;
            default: next_state = state_1;
        endcase
    end

endmodule
```

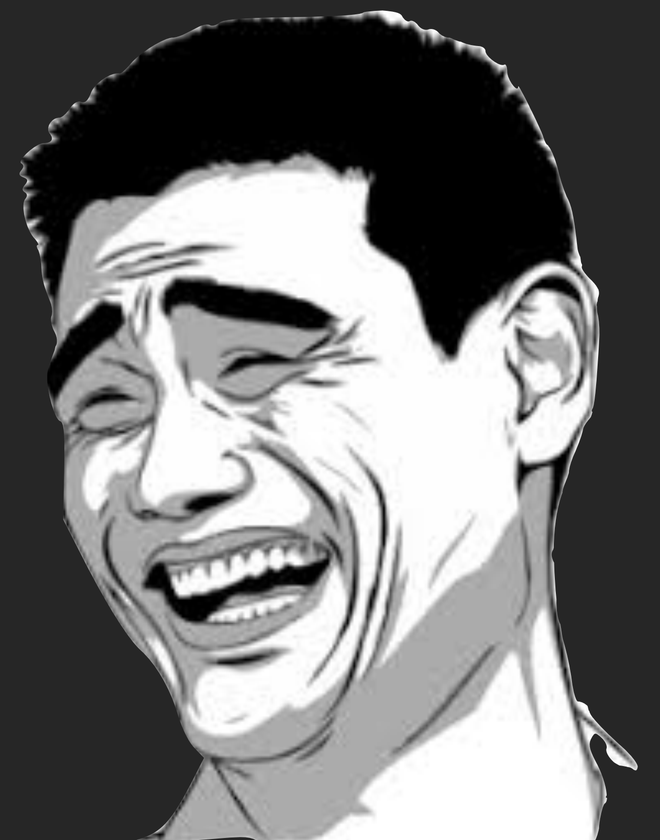
Q & A

Q & A

- 第一題
  - RTL 的全名是？？

Q & A

- 第二題
  - 這是 ppt 的第幾頁？



Outline

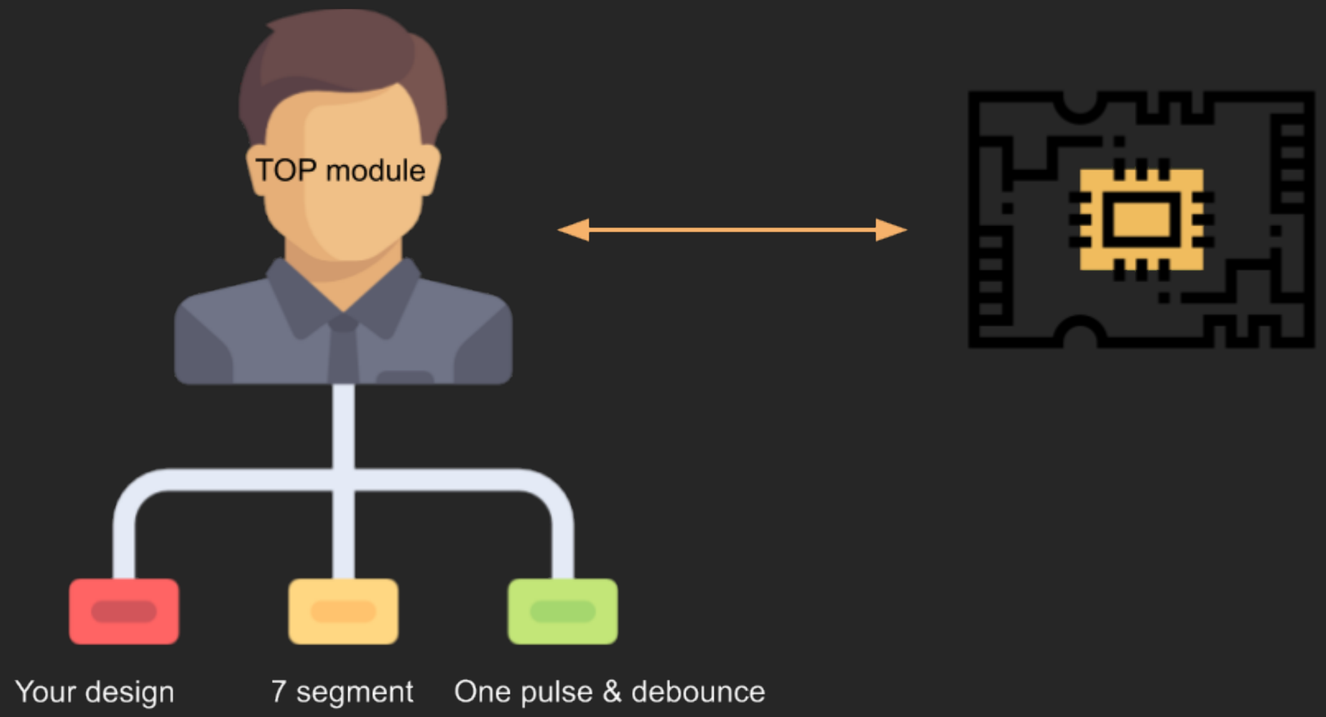
1 Code Structure

2 Coding Style

3 **Module Relation**

4 Useful Module

Module Relation (1/3)



Module Relation (2/3)

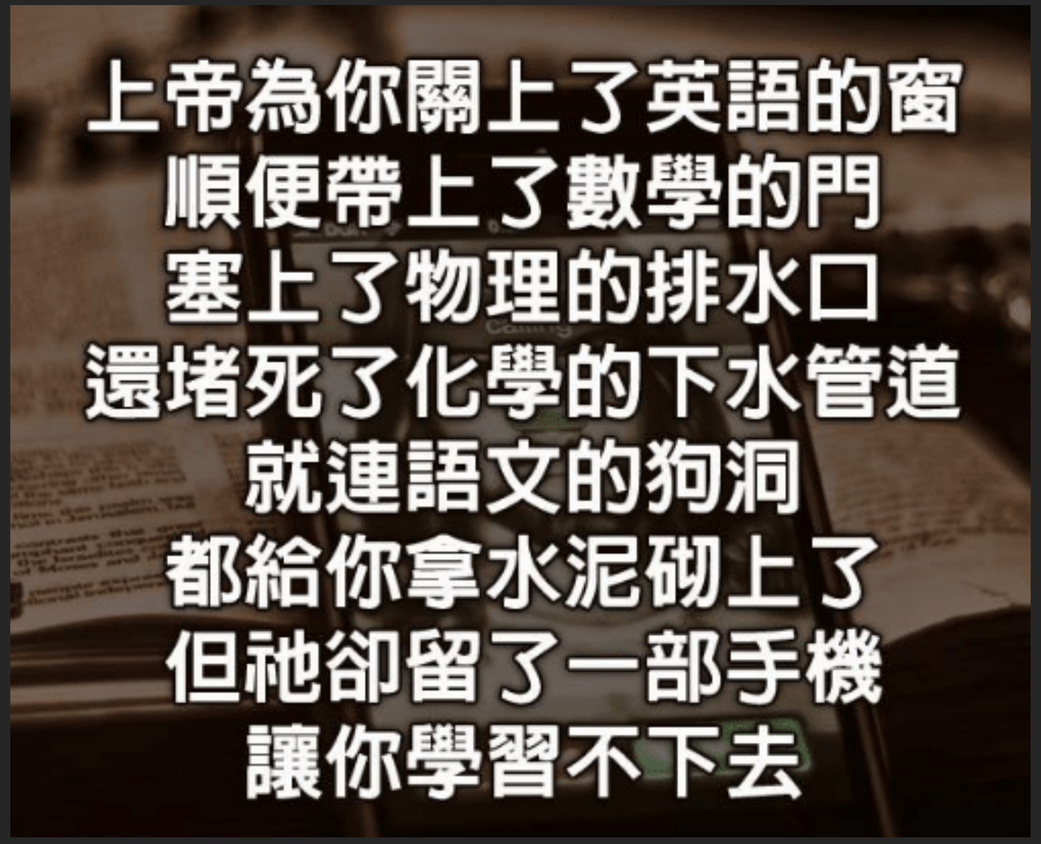
```
module Parameterized_Ping_Pong_Counter (CLK100MHZ, sw, btnU, btnD, seg, dp, an);

    input CLK100MHZ;
    input [16-1:0] sw;
    input btnU, btnD;
    output [7-1:0] seg;
    output [4-1:0] an;
    output dp;

    wire observable_clk, clk, u, d, U, D;
    wire direction;
    wire [4-1:0] out;

    debounce Uppp(.pb_debounced(U), .pb(btnU), .clk(CLK100MHZ));
    debounce Dooo(.pb_debounced(D), .pb(btnD), .clk(CLK100MHZ));

    onepulse Up(.in(U), .dclk(CLK100MHZ), .out(u));
    onepulse Do(.in(D), .dclk(CLK100MHZ), .out(d));

    Counter count(.CLK100MHZ(CLK100MHZ), .RESET(d), .enable(sw[0]), .max(sw[8:5]), .min(sw[4:1]), .flip(u), .direction(direction), .out(out));

    FPGA_Seven_Segment sv(.CLK100MHZ(CLK100MHZ), .RESET(1'b0), .direction(direction), .out(out), .seg(seg), .an(an), .dp(dp));

endmodule
```

Module Relation (3/3)

- Block Diagram
  - The relationship between modules
  - The signals flow

Q & A

Q & A



- 第一題
  - Top module 所扮演的角色？

Q & A

- 第二題
  - 下一個章節叫什麼？

Outline

1 Code Structure

2 Coding Style

3 Module Relation

4 **Useful Module**

Useful Module (1/4)

- Clock Divider

```
95  module Clock_Divider #(parameter n=25) (CLK, RESET, dclk);
96      input CLK, RESET;
97      output dclk;
98
99      reg [n-1:0] num;
100     wire [n-1:0] next_num;
101
102     always @(posedge CLK) begin
103         if(RESET)   num <= 0;
104         else        num <= next_num;
105     end
106
107     assign next_num = num + 1'b1;
108     assign dclk = num[n-1];
109 endmodule
110
```

Useful Module (2/4)

- 7 segment display

```
module FPGA_Seven_Segment (CLK100MHZ, RESET, value, seg, an);

    input CLK100MHZ, RESET;
    input [16-1:0] value;
    output reg [7-1:0] seg;
    output reg [4-1:0] an;
    reg [3-1:0] state, next_state;
    wire[7-1:0] seg0, seg1, seg2, seg3;
    wire clk;

    parameter state1 = 2'b00;
    parameter state2 = 2'b01;
    parameter state3 = 2'b10;
    parameter state4 = 2'b11;

    clock_divider #(13) t(.CLK(CLK100MHZ), .RESET( RESET), .dclk(clk));

    display A0 (.sw(value[ 3: 0]), .seg(seg0));
    display A1 (.sw(value[ 7: 4]), .seg(seg1));
    display A2 (.sw(value[11: 8]), .seg(seg2));
    display A3 (.sw(value[15:12]), .seg(seg3));

    always @(posedge clk)begin
        if(RESET)   state   <= state1;
        else        state   <= next_state;
    end

    always @(posedge clk)begin
        case(state)
            state1:begin
                seg         <= seg0;
                an          <= 4'b1110;
                next_state  <= state2;
            end
            state2:begin
                seg         <= seg1;
                an          <= 4'b1101;
                next_state  <= state3;
            end
            state3:begin
                seg         <= seg2;
                an          <= 4'b1111;
                next_state  <= state4;
            end
            state4:begin
                seg         <= seg3;
                an          <= 4'b1111;
                next_state  <= state1;
            end
        endcase
    end
endmodule

```

Useful Module (3/4)

```
54  module display (sw, seg);
55      input [3:0] sw;
56      output reg [6:0] seg;
57
58      always@(*)begin
59      case(sw)
60          4'd0:    seg = 7'b1000000;
61          4'd1:    seg = 7'b1111001;
62          4'd2:    seg = 7'b0100100;
63          4'd3:    seg = 7'b0110000;
64          4'd4:    seg = 7'b0011001;
65          4'd5:    seg = 7'b0010010;
66          4'd6:    seg = 7'b0000010;
67          4'd7:    seg = 7'b1011000;
68          4'd8:    seg = 7'b0000000;
69          4'd9:    seg = 7'b0010000;
70          4'b1010:seg = 7'b0001000;
71          4'b1011:seg = 7'b1100000;
72          4'b1100:seg = 7'b0110001;
73          4'b1101:seg = 7'b1000010;
74          4'b1110:seg = 7'b0110000;
75          4'b1111:seg = 7'b0111000;
76      endcase
77      end
78  endmodule
79
80
```

Docker

- 7 segment display

Useful Module (4/4)

- Counter

```
82  module counter #(parameter n=8) (CLK, RESET, count);
83      input CLK, RESET;
84      output  count;
85      reg [n:0] i;
86      wire[n:0]next_i;
87          always @(posedge CLK) begin
88          if(RESET)   i <= 0;
89          else        i <= next_i;
90      end
91
92      assign next_i = i + 7'b1;
93      assign count = (&(i)) ? 1'b1 : 1'b0;
94  endmodule
95
96
97
```

End